



# Week7: Improving Networks with Regularization

SETH HUANG

# THERE ARE SEVERAL BASIC WAYS TO IMPROVE NETWORKS (AKA: REDUCE OVERFITTING):

- Increase the amount of data (can often be done artificially)
- Reduce the “size” of our network: BUT bigger networks are potentially more powerful
- Regularization

# WEIGHT DECAY OR L2 REGULARIZATION

$$\frac{\lambda}{2n} \sum_w w^2$$

$\lambda/2n$  :

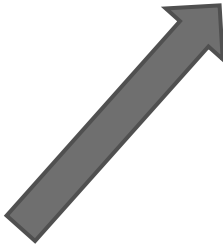
- This is a “regularization parameter”
- And  $\lambda > 0$
- Note that it does not include biases

# REGULARIZATION: WEIGHT DECAY OR L2 REGULARIZATION

The basic idea is to change the cost function just a little bit.

$$C = -\frac{1}{n} \sum_{x_j} \left[ y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L) \right] + \frac{\lambda}{2n} \sum_w w^2.$$

Cross-Entropy



Extra Term (??)



Make sure you recall what this cost function is.

## WEIGHT DECAY/ L2 REGULARIZATION:

$$\frac{\lambda}{2n} \sum_w w^2$$

$$\lambda/2n$$

This is a regularization parameter. Lambda is always  $> 0$

# THE INTUITION

Make the network “prefer” to learn “small weights” instead of “large weights”

Large weights are only allowed if they improve (lower the cost function) learning dramatically.

$$C = -\frac{1}{n} \sum_{x_j} \left[ y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L) \right] + \frac{\lambda}{2n} \sum_w w^2.$$

## LET'S DIG A BIT DEEPER ABOUT WHY IT HELPS

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

The first term,  $C_0$ , is the original cost function.

When  $\lambda$  is small, we prefer to minimize to original function

But why and how?

# EXAMINING THE PARTIAL DERIVATIVES

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}.$$



The updating rule



$$\begin{aligned} w &\rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} w \\ &= \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}. \end{aligned}$$



BEFORE WE GET LOST, REMEMBER L2  
REGULARIZATION IS ALSO CALLED “WEIGHT  
DECAY”

$$\begin{aligned}w &\rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} w \\&= \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}.\end{aligned}$$

Notice now, with the first term, the weights have been rescaled!

This rescaling makes the weight smaller. And notice the first term makes the weight smaller EACH TIME it's updated. That is why it's called “weight decay.”

Does that mean the weights will become zero?! The universe is all but full of empty weights. Check the second term, what do you see?

# BASICALLY, THE UPDATING RULE BECOMES A HORSE RACE!

Particular weights only get bigger if, by changing them, SIGNIFICANTLY reduce THE COST.

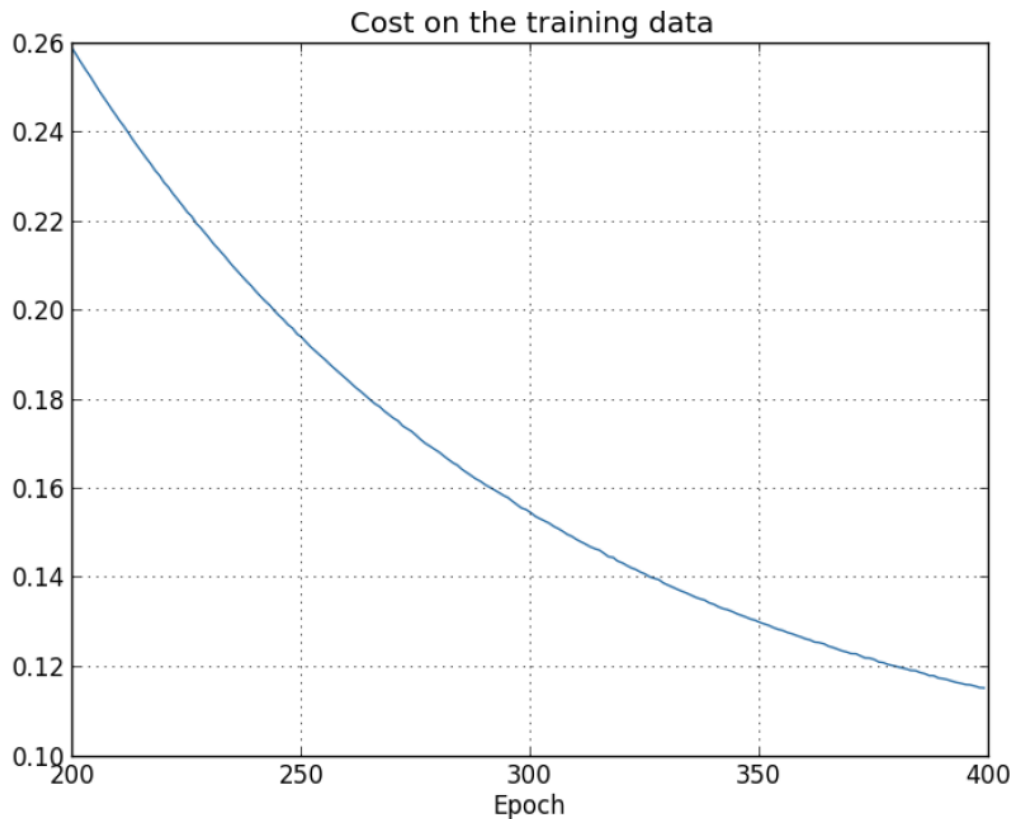
HOW SIGNIFICANT? It's determined by Lambda



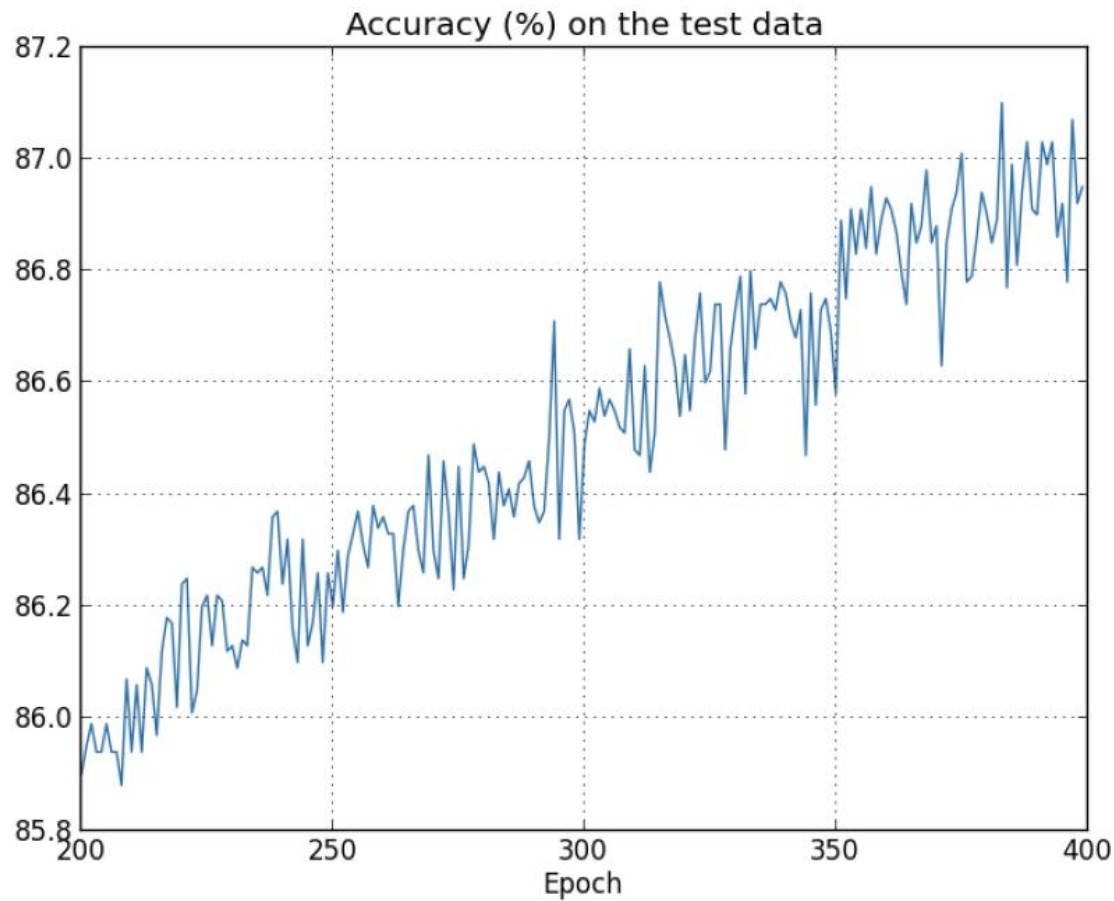
# NOW, TRY IT:

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data[:1000], 400, 10, 0.5,
... evaluation_data=test_data, lambda = 0.1,
... monitor_evaluation_cost=True, monitor_evaluation_accuracy=True,
... monitor_training_cost=True, monitor_training_accuracy=True)
```

YOU SHOULD SEE THIS (GRAPHICALLY):  
CAN YOU GET ANY INSIGHT FROM THIS?



# NOW, THIS IS THE REAL DEAL

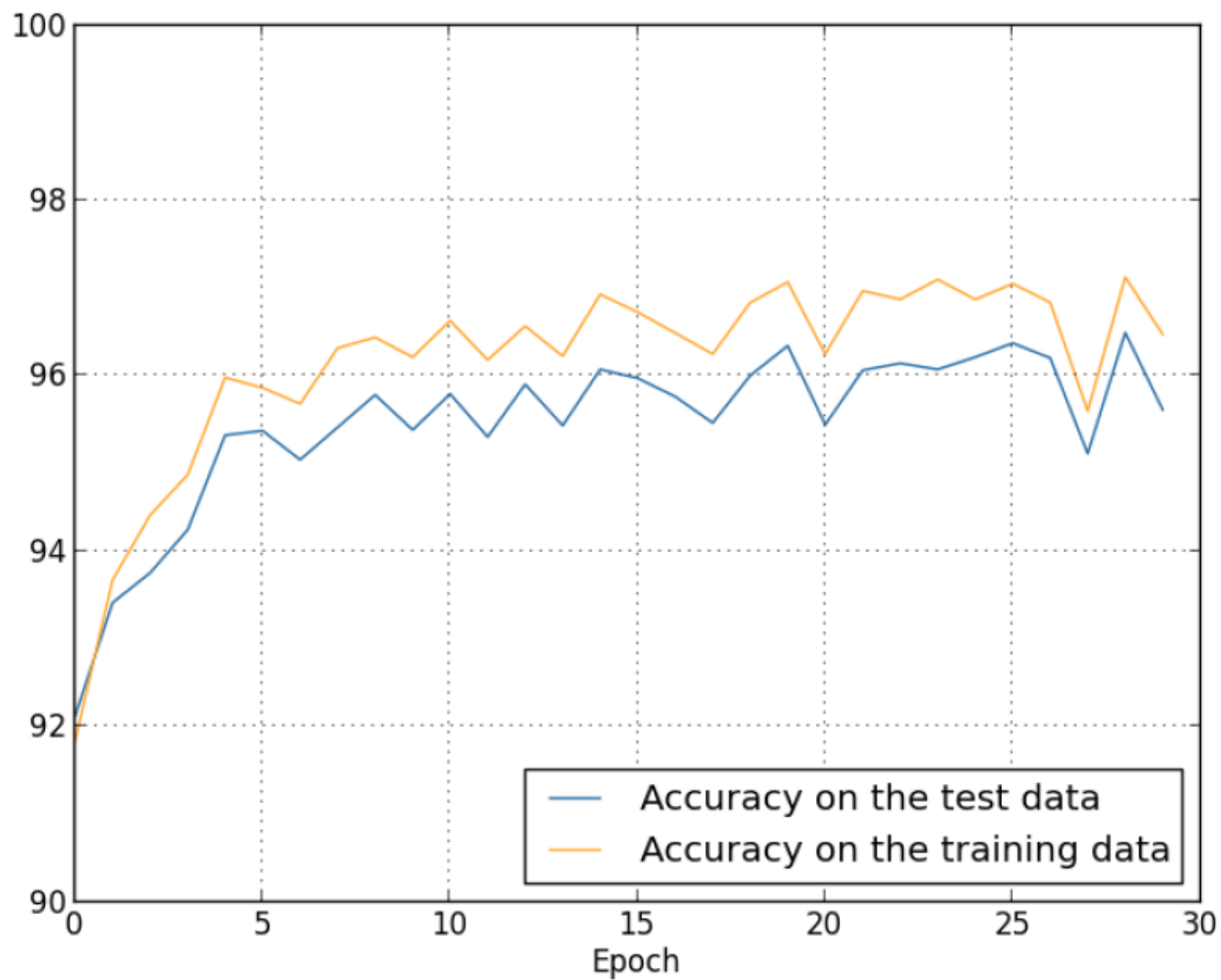


NOTICE HOW MANY OBSERVATIONS FROM  
YOUR TRAINING DATA YOU ARE TRAINING WITH?

```
>>> net.large_weight_initializer()  
>>> net.SGD(training_data, 30, 10, 0.5,  
... evaluation_data=test_data, lmbda = 5.0,  
... monitor_evaluation_accuracy=True, monitor_training_accuracy=True)
```

But you need to change the regularization parameter as well.

$$\begin{aligned} w &\rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} w \\ &= \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}. \end{aligned}$$



# YOUR TURN: GO TO NETWORK2 AND FIND WHERE L2 REGULARIZATION IS APPLIED

1. Find where it is applied in network2
2. Use the print function so that, when your program is running, it will say "I am regularized!"



# WHY DOES REGULARIZATION REDUCE OVERFITTING? WHAT IS THE STORY HERE?

Long, long time ago in a galaxy far far away...

A given connection have lower weights

⇒ Some connections are disregarded

⇒ A network on a “weight control”

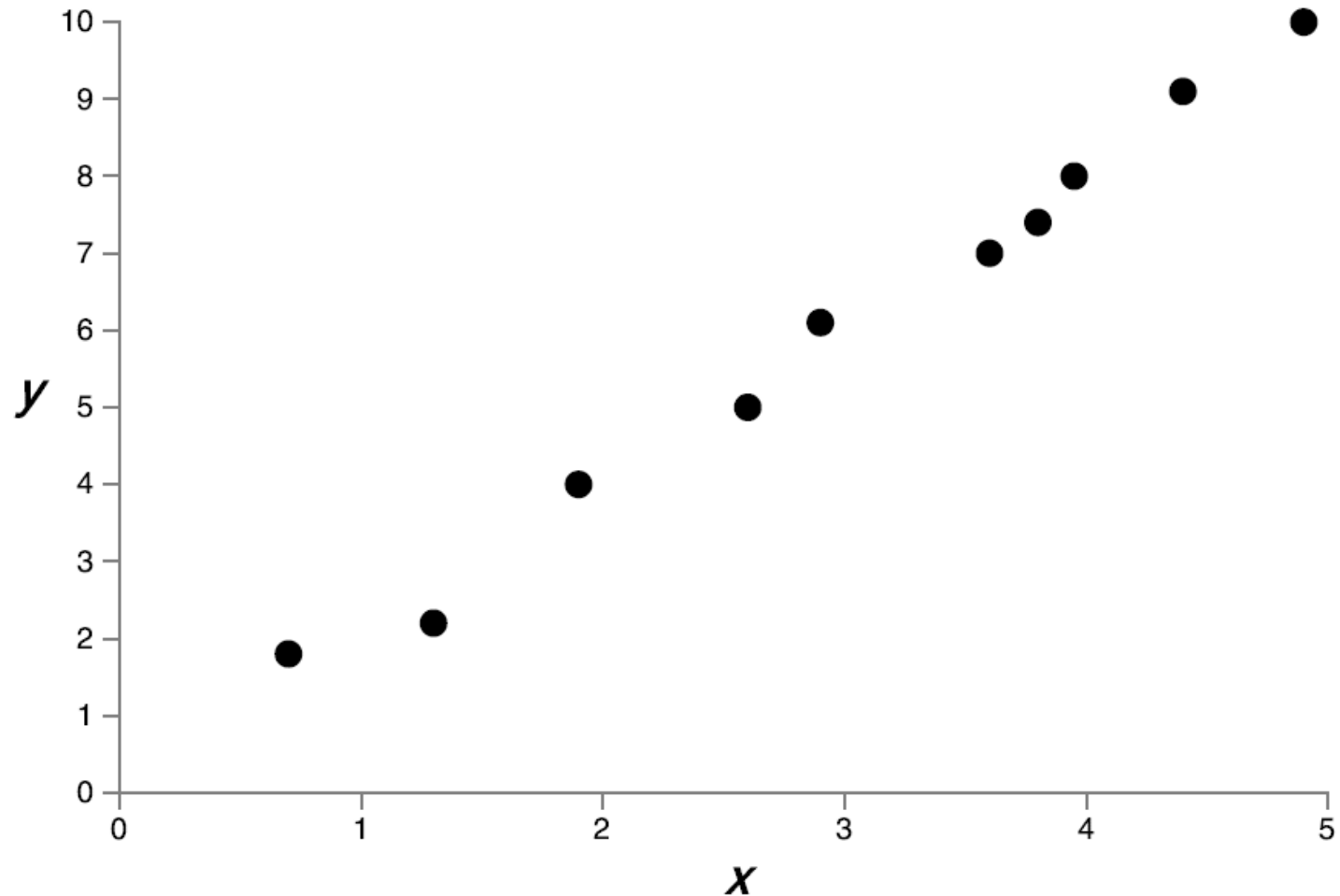
⇒ Lower “complexity”

⇒ Doesn't overfit too much

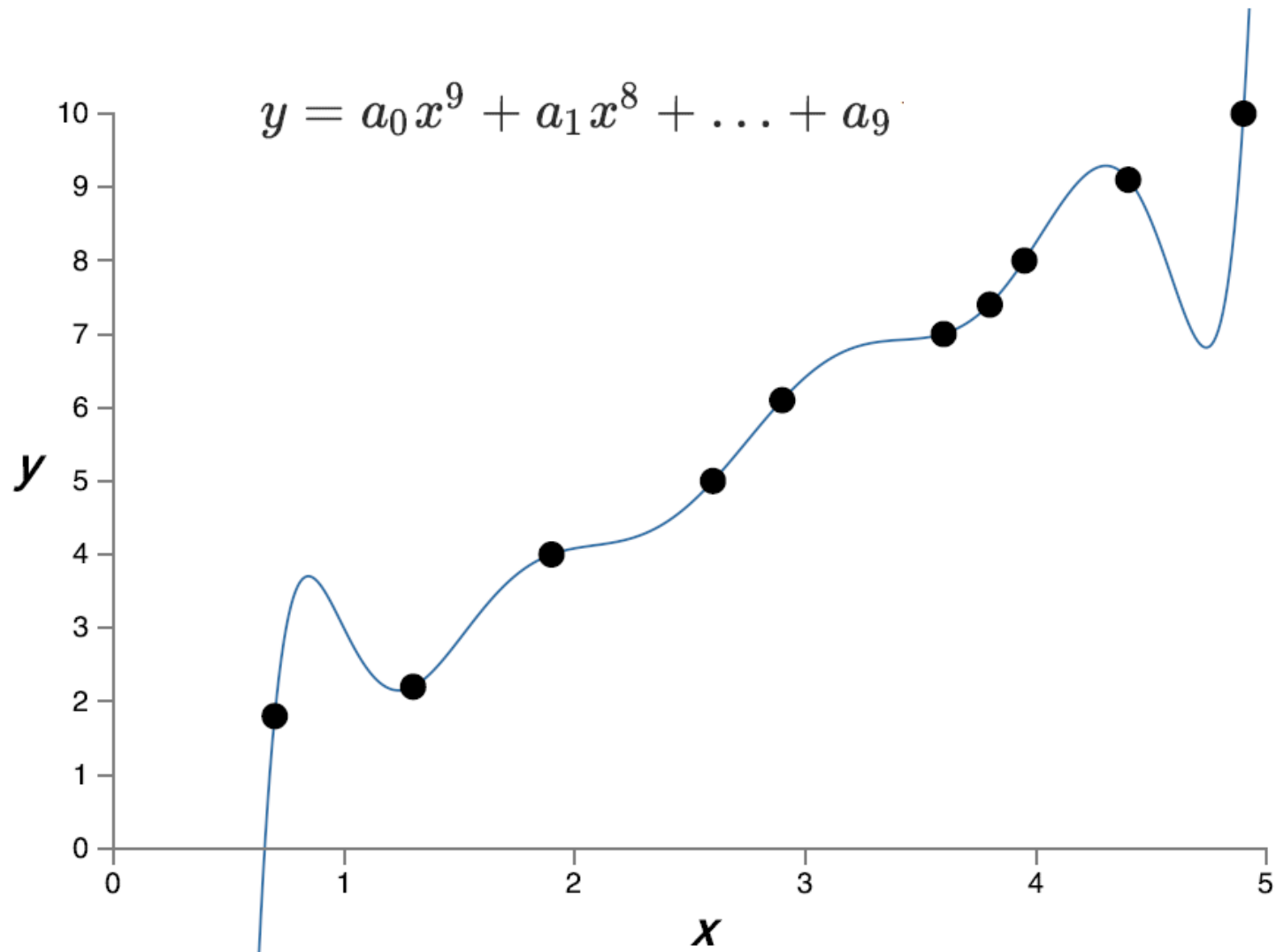
At least, that is the logic.

WHY REGULARIZATION  
WORKS?

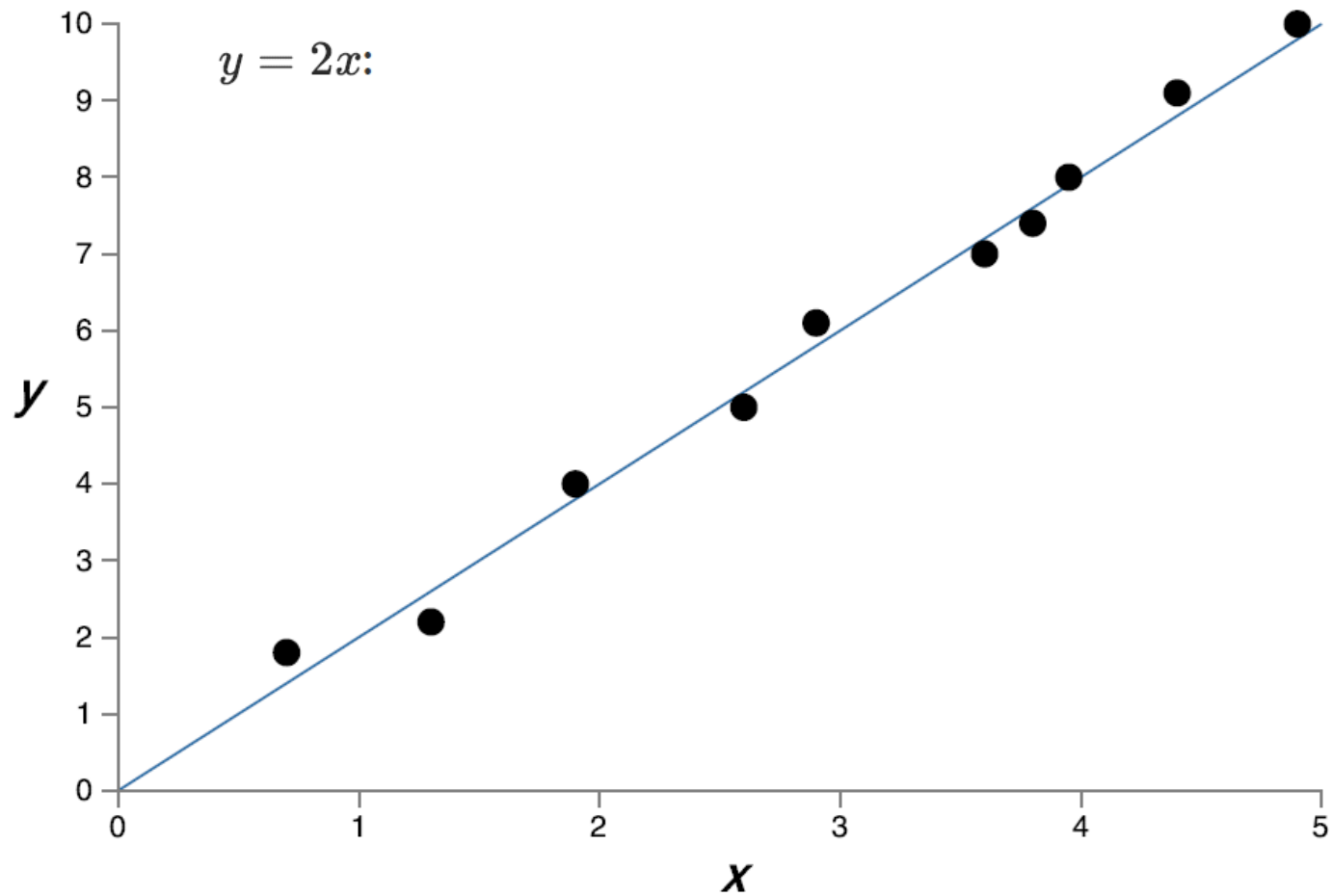
EXAMINING A SIMPLE DATA SET. YOU  
ARE TRYING TO FIT YOUR MODEL



# YOU CAN FIT THE MODEL THIS WAY



OR THIS WAY



# WHICH MODEL IS BETTER? WHICH MODEL SHOULD GENERALIZE BETTER?

People tend to use “rule of thumbs” to describe the world events, but we cannot answer that until we know the underlying real-world structure.

There are two possibilities:

1. The complex model truly describes the world (for example, what if your broken shoelace actually determines the luck of the day or is a precursor of it?
2. The simple model addresses the noises and measurement error, so it is okay if the model does not fit perfectly.

How does it relate to our neural networks?

# YES, SIMPLER MODEL MAY HAVE MORE PREDICTIVE POWER, BUT AVOID TRYING OVERSIMPLIFYING THINGS (SUCH AS MAKING MODELS SMALLER)

A few key points:

1. The smallness of weights in general means that the prediction values will not change dramatically even if there are noises in the data.



# SIMPLER MODEL MAY HAVE MORE PREDICTIVE POWER (GENERALLY SPEAKING)

A few key points:

2. Large weights are sensitive to small changes in output, and will behave drastically.

An un-regularized network can use large weights to learn a complex model that carries a lot of information about the noise in the training data

In a nutshell, **regularized networks are constrained to build relatively simple models based on patterns seen often in the training data**, and are resistant to learning peculiarities of the noise in the training data.



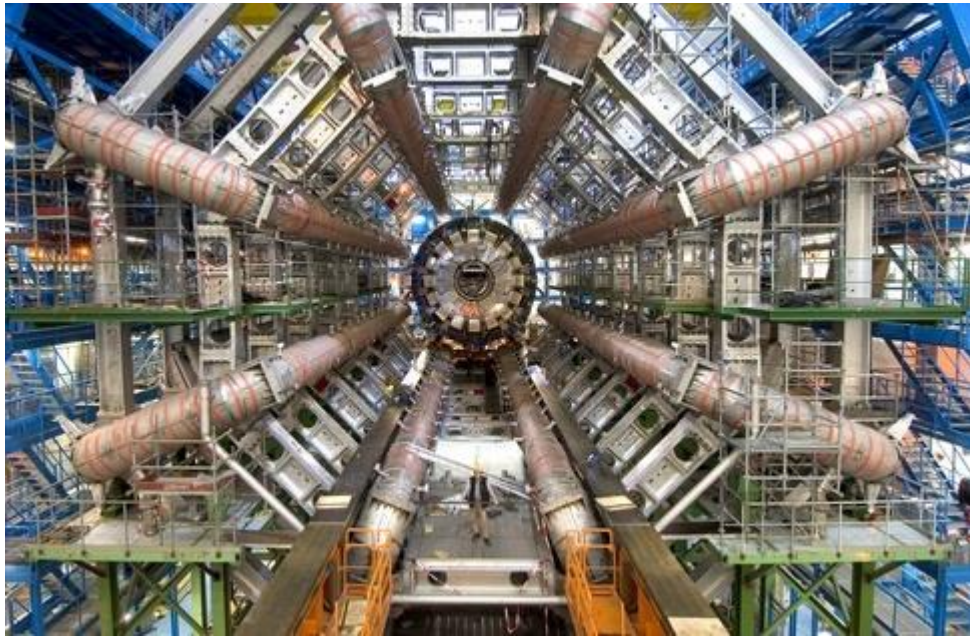
SO, SIMPLER IS BETTER, RIGHT?



**KEEP  
CALM  
AND USE  
OCCAM'S  
RAZOR**

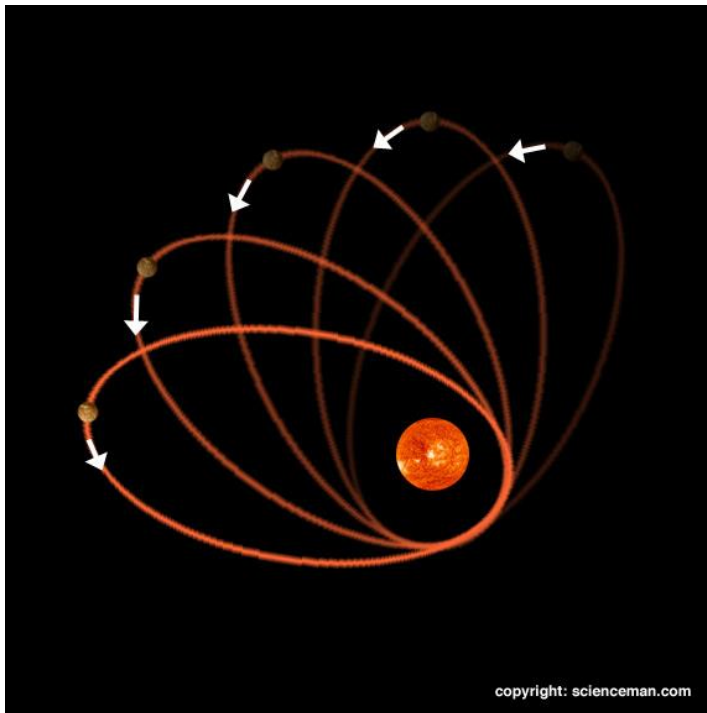
# OCCAM'S RAZOR

There is no *a priori* logical reason to prefer simple explanations over more complex explanations. Indeed, sometimes the more complex explanation turns out to be correct.



# OCCAM'S RAZOR IS SOMETIMES WRONG IN SCIENCE

For example, the orbit of Mercury does not exhibit the Shape as predicted by Newton physics. Einstein provided an alternative theory that is much more mathematically complex but proven to be right.



# WHAT ARE THE MORALS OF THE DISCUSSION THEN?

1. It is really not easy to judge, when both theories can be used to explain an event, the simpler one is always the preferred one.
2. The virtue of a model is not its simplicity but its applicability.

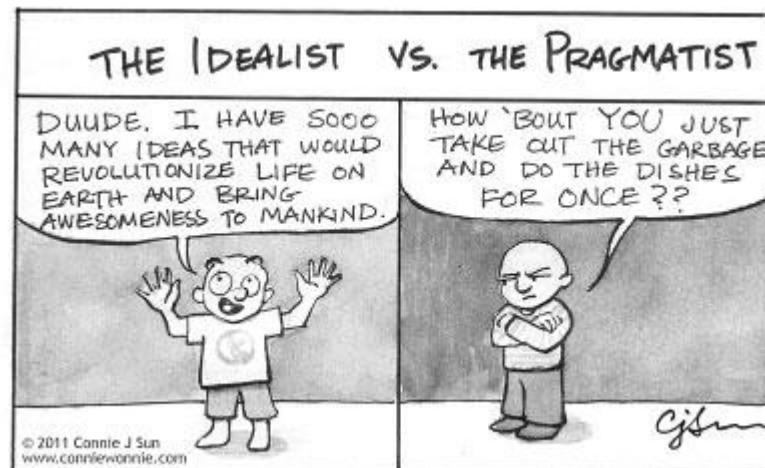
Fat Tony thought exercise:

When you toss a coin, and you get 17 heads out of 20 tosses. What would you be thinking?

WHICH MEANS, IN TERMS OF NEURAL NETWORKS, REGULARIZED ALMOST ALWAYS PERFORM BETTER THAN NON-REGULARIZED ONES.

=> USE IT!

While it often helps, we don't have an entirely satisfactory systematic understanding of what's going on, merely incomplete heuristics and rules of thumb.



REGULARIZATION GIVES AS AN EXAMPLE ON A TECHNIQUE OF HOW A NETWORK CAN BE IMPROVED, BUT IT DOES NOT INDICATE IT'S HOW GENERALIZATION WORKS:

YOUR MODEL WILL WORK (IF A GOOD ONE) UNTIL IT CEASES TO WORK.

# IT IS SOMETIMES CALLED THE PROBLEM OF INDUCTION BY DAVID HUME

1. You have observed all white swans, and you generalize the observations (the property of the swan class) that all swans are white.
2. The sequence of events that have happened in the past will continue to happen the same way

# IT IS SOMETIMES CALLED THE PROBLEM OF INDUCTION BY DAVID HUME

1. You have observed all white swans, and you generalize the observations (the property of the swan class) that all swans are white.
2. The sequence of events that have happened in the past will continue to happen the same way




# OTHER REGULARIZATION TECHNIQUES

- L1 Regularization
- Dropout
- Artificially expand the training data

# L1 REGULARIZATION

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|.$$


$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} \text{sgn}(w)$$



COMPARED WITH

$$w \rightarrow w' = w - \frac{\eta\lambda}{n} \text{sgn}(w) - \eta \frac{\partial C_0}{\partial w},$$

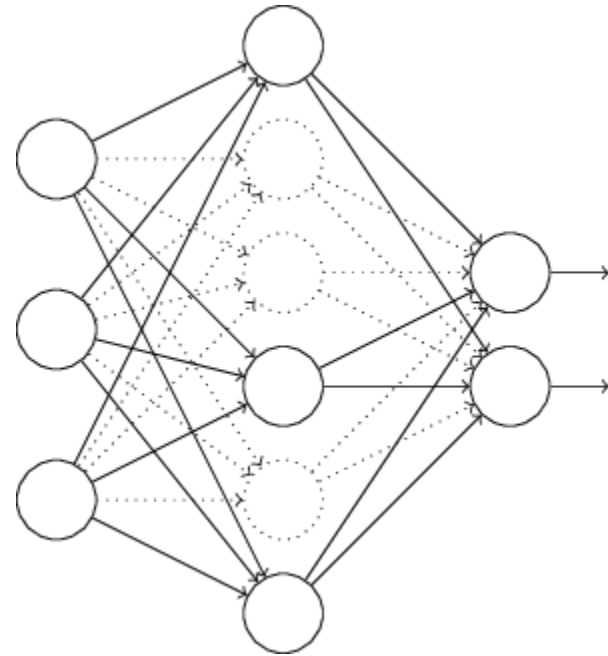
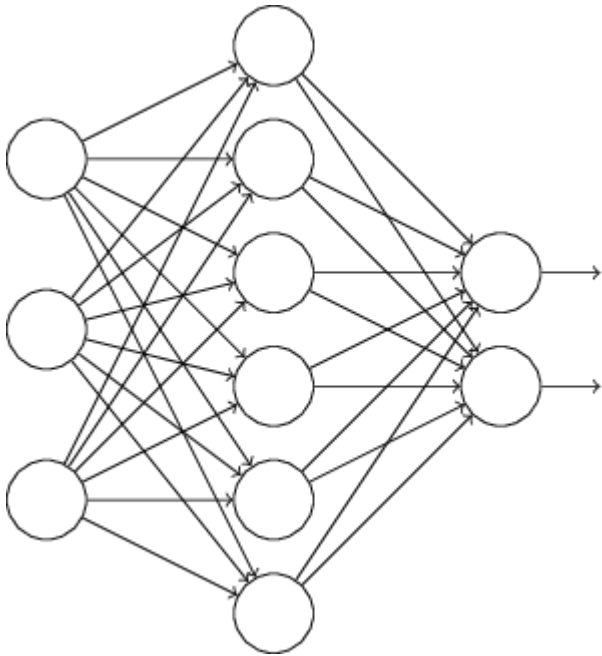
$$w \rightarrow w' = w \left( 1 - \frac{\eta\lambda}{n} \right) - \eta \frac{\partial C_0}{\partial w}.$$

$\text{sgn}(w) = 1$  if  $w$  is positive

$\text{sgn}(w) = -1$  if  $w$  is negative

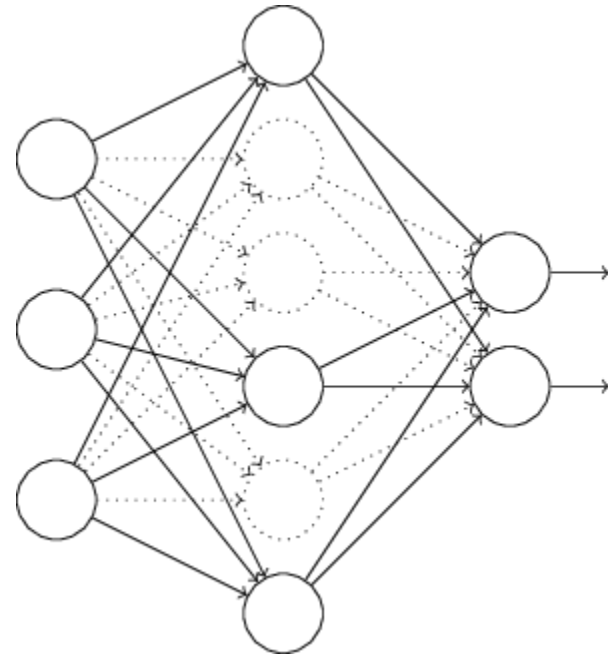
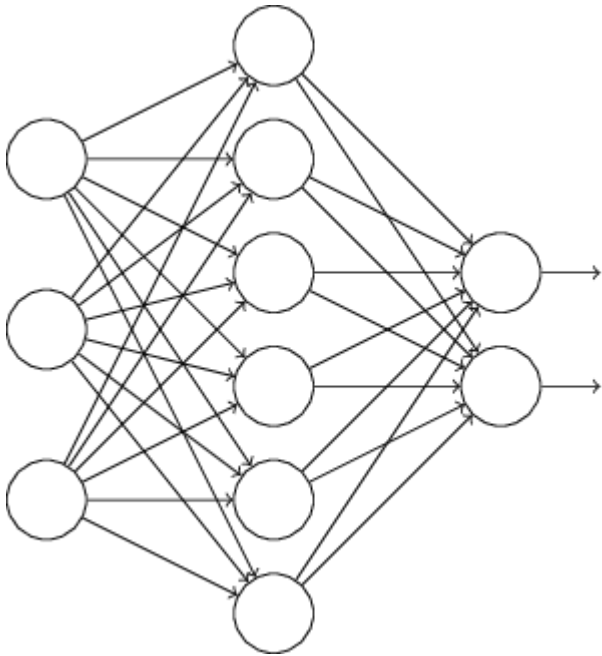
# DROP OUT: NOT CHANGING THE FUNCTION BUT THE ENTIRE NETWORK

Drop out is a common practice in deep learning and is important. We will use it in later chapters.



# DROP OUT

We start by randomly deleting some hidden neurons for one mini-batch, and then we do the training. And then randomly delete some hidden neurons (half of them in this case), and repeat.



You will still have the same number of parameters in the end, but the weights of them are a bit different.

# DROPOUT IS WEIRD, YES

Imagine you have trained several networks. When that happens we could use some kind of averaging or voting scheme to decide which output to accept. For instance, if we have trained five networks, and three of them are classifying a digit as a "3", then it probably really is a "3". The other two networks are probably just making a mistake.

This is actually a powerful (but expensive) technique to reduce overfitting.

Why does voting work? The reason is that the different networks may overfit in different ways, and averaging may help eliminate that kind of overfitting.

# FOR NOW, WE WILL STICK TO L2 REGULARIZATION UNTIL WE REACH DEEP LEARNING (VERY SOON)

Try the overfitting codes. Can you get it to run?

```
9 import numpy as np
0
1
2 def main(filename, num_epochs,
3         training_cost_xmin=0,
4         test_accuracy_xmin=0,
5         test_cost_xmin=0,
6         training_accuracy_xmin=0,
7         training_set_size=1000,
8         lambda=0.0):
9     """`filename` is the name of the file where the results will be
0     stored. `num_epochs` is the number of epochs to train for.
1     `training_set_size` is the number of images to train on.
2     `lambda` is the regularization parameter. The other parameters
3     set the epochs at which to start plotting on the x axis.
4     """
5     run_network(filename, num_epochs, training_set_size, lambda)
6     make_plots(filename, num_epochs,
7               test_accuracy_xmin,
8               training_cost_xmin,
9               test_accuracy_xmin,
0               training_accuracy_xmin,
1               training_set_size)
2
```